5

METHOD AND APPARATUS FOR

10

SUPPORTING READ-ONLY OBJECTS WITHIN

AN OBJECT-ADDRESSED MEMORY

HIERARCHY

15   **Inventors:** Matthew L. Seidl, Gregory M. Wright and Mario I. Wolczko

BACKGROUND

20   **Field of the Invention**

[0001] The present invention relates to the design of computer systems that support efficient references to objects defined within an object-oriented programming system. More specifically, the present invention relates to a method and an apparatus for supporting read-only objects within an object-addressed

25   memory hierarchy.

Related Art

[0002] As object-oriented programming languages become more widely used, computer systems are being designed to manipulate objects more efficiently.

1

In particular, some proposed object-addressed memory hierarchies include object-addressed caches that facilitate lookups based on location-independent object identifiers (OIDs) without having to first translate the OIDs into corresponding physical addresses. Furthermore, during cache miss operations, these object-

5   addressed memory hierarchies also support translations between OIDs (used to access copies of objects in the object-addressed caches) and corresponding physical addresses (used to access copies of the objects in main memory).

[0003] Unfortunately, these proposed object-addressed memory hierarchies do not provide support for read-only copies of objects. This makes it

10   hard to incorporate such object-addressed memory hierarchies into modern distributed object systems, which allow read-only copies of an object to reside on multiple hosts. Allowing read-only copies of an object to reside on multiple hosts provides significant performance advantages, because it enables each host to access a read-only copy of the object in its local cache, so long as the host only

15   performs read operations on the object. Note that these local cache accesses can be orders of magnitude faster than non-local accesses.

[0004] Hence, what is needed is a method and an apparatus that supports a read-only copy of an object within a distributed object-addressed memory hierarchy.

20

SUMMARY

[0005] One embodiment of the present invention provides a system that supports read-only objects within an object-addressed memory hierarchy. During operation, the system receives a request to access an object, wherein the request

25   includes an object identifier for the object that is used to reference the object within the object-addressed memory hierarchy. In response to this request, the system uses the object identifier to retrieve an object table entry associated with

2

the object. If the request is a write request, the system examines a read-only indicator within the object table entry. If this read-only indicator specifies that the object is a read-only object, the system performs a corrective action to deal with the fact that the write request is directed to a read-only object.

5   [0006] In a variation on this embodiment, if the request is a read request, the system uses a physical address from the object table entry to access the object in main memory.

[0007] In a variation on this embodiment, performing the corrective action can involve causing a fault handler in the requesting processor to perform the

10   corrective action.

[0008] In a variation on this embodiment, performing the corrective action can involve obtaining a writable copy of the object, clearing the read-only indicator to indicate that the object is no longer read-only, and updating the writable copy of the object with data from the write request. It can also involve

15   updating a remotely located master copy of the object with data from the write request. The corrective action can alternatively involve terminating the requesting process because the write request is not allowed. Furthermore, if the request is directed to a debugging breakpoint (or watch point), the corrective action can involve pausing the requesting process at the debugging breakpoint and clearing

20   the read-only indicator.

[0009] In a variation on this embodiment, the request to access the object is received at a translator that translates between object identifiers (used to reference objects in an object cache) and physical addresses (used to address objects in main memory).

25   [0010] In a further variation, prior to receiving the request at the translator, the request is initially directed to the object cache. If the request causes a hit in the object cache, the object is accessed in the object cache and the request is not

3

sent to the translator. Otherwise, if the request causes a miss in the object cache, the request is sent to the translator.

[0011] In a variation on this embodiment, the system makes a given object read-only by setting a read-only indicator associated with the given object to indicate that the given object is read-only and then causing all object caches within a local cache-coherent domain to flush any modified cache lines for the given object out to main memory. In this way, subsequent upgrades of the given object from read-only status to writable or modified status in any caches in the local cache-coherent domain must go through a translator.

[0012] In a further variation, the system causes all object caches within the local cache-coherent domain to flush any modified cache lines for the given object out to main memory by executing read-with-intent-to-only-read (RWITOR) instructions on each cache line of the given object.

[0013] In a further variation, the system makes the given object read only in response to a request received from outside the local cache-coherent domain.

[0014] In a further variation, the translator includes hardware to translate between object identifiers and physical addresses.

BRIEF DESCRIPTION OF THE FIGURES

[0015] FIG. 1 illustrates a computer system in accordance with an embodiment of the present invention.

[0016] FIG. 2 illustrates an object table entry in accordance with an embodiment of the present invention.

[0017] FIG. 3 presents a flow chart illustrating the process of using an object identifier to access an object in accordance with an embodiment of the present invention.

4

[0018] FIG. 4 presents a flow chart illustrating the process of executing a read-with-intent-to-only-read (RWITOR) instruction in accordance with an embodiment of the present invention.

[0019] FIG. 5 presents a flow chart illustrating the processing of a foreign read-to-own (RTO) signal received from a remote machine in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

[0020] The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

Computer System

[0021] FIG. 1 illustrates a computer system 100 in accordance with an embodiment of the present invention. Computer system 100 can generally include any type of computer system, including, but not limited to, a computer system based on a microprocessor, a mainframe computer, a digital signal processor, a portable computing device, a personal organizer, a device controller, and a computational engine within an appliance.

[0022] As is illustrated in FIG. 1, computer system 100 includes processors 102 and 103. Processors 102 and 103 access code and data from L1

5

caches 104 and 105, respectively. Note that L1 caches 104 and 105 can include unified instruction/data caches, or alternatively, separate instruction caches and data caches.

[0023] Processors 102 and 103 are associated with translation lookaside buffers (TLBs) 114 and 115, which facilitate translating virtual addresses into physical addresses for non-object references.

[0024] L1 cache 104 and L1 cache 105 make use of an extended address encoding procedure that enables L1 cache 104 to function as both a conventional cache and an object cache. For example, during a conventional load operation, a virtual address is sent from processor 102 to TLB 114. TLB 114 translates the virtual address into a physical address, which is subsequently used by L1 cache 104.

[0025] In contrast, during a load operation involving a portion of an object, processor 102 obtains the corresponding object ID (OID) and offset and combines them to create an object address. This object address is embedded into an unused portion of the physical address space to produce an encoded address. Note that the higher order bits of this encoded address are typically different than the higher order bits of any physical address. This allows the system to distinguish an encoded address from a physical address. When the encoded address is subsequently sent from processor 102 to L1 cache 104, the encoded address bypasses TLB 114 and directly enters L1 cache 104. Note that only minor modifications are required to conventional cache designs in order to provide object caching using the above-described technique.

[0026] In order to request a non-object data item, such as a value from a normal virtual address, processor 102 generates a virtual address that is sent to TLB 114. TLB 114 translates this virtual address into a physical address, which is sent to L1 cache 104.

6

[0027] Note that after an object address is translated into an encoded address L1 cache 104, L1 cache 105 and L2 cache 106 can treat the encoded address in the same manner as a normal physical address.

[0028] If a given data item (or instruction) is not located within L1 cache 104 or L1 cache 105, it is retrieved from L2 cache 106. If it is not located within L2 cache 106, it is pulled into L2 cache 106 from main memory 110.

[0029] Unlike in a conventional memory hierarchy, a translator 108 is interposed between L2 cache 106 and main memory 110. Translator 108 converts an object address, comprising an object ID and an offset, into a corresponding physical address, which is sent to main memory 110.

[0030] If an object is not present within L2 cache 106, the encoded address is forwarded to translator 108. Translator 108 uses an object table 109 to translate the encoded address into a corresponding physical address. Each entry in object table 109 associates a given object ID with a corresponding physical address in main memory where the object resides. (Note that in one embodiment of the present invention, object table 109 is implemented as a table that resides in main memory 110.)

[0031] When a cache miss for an object occurs in L2 cache 106, translator 108 intercepts the encoded address and extracts the object ID. Next, translator 108 uses the object ID to index into the object table 109 for a corresponding physical address. Once the physical address is found, translator 108 converts the load request for the object into a load request for a physical address in main memory 110.

[0032] The system uses the physical address and the offset to locate a specific cache line (or cache lines) in main memory 110. Fetching circuitry within translator 108 issues a load request to main memory 110. This fetching circuitry

7

subsequently receives the cache line corresponding to the physical address. The fetching circuitry then forwards the cache line to L2 cache 106.

[0033] Object cache lines differ from conventional physical cache lines because object cache lines can start on arbitrary word boundaries, whereas physical cache lines are delineated by larger power-of-two address boundaries. Hence, physical cache lines and object cache lines may not always align. For example, a physical cache line with a length of 64 bytes typically starts at a physical address that is a multiple of 64. Objects, however, may start on any physical address which is a multiple of four in a 32-bit system. Thus, a 64-byte object cache line starting at address 44 includes addresses (44..107). This overlaps with physical cache lines (0..63) and (64..127). In this case, the object is split across two physical cache lines. Hence, two memory read operations are required to retrieve the entire object cache line. Once both physical cache lines have been retrieved, the portions of the cache lines containing the object cache line, (44..63) and (64..107), are concatenated together to form the object cache line (44..107). Other portions of the physical cache lines are discarded.

[0034] In the event of an eviction from L2 cache 106, translator 108 converts the encoded address containing the object ID and the offset into a physical address. The fetching circuitry subsequently uses the physical address to generate a store operation to store the evicted cache line in main memory 110. Note that during the process of evicting an object line, the system may have to perform read-modify-write operations on two physical cache lines.

[0035] Note that processors 102 and 103 are configured to handle the extended address encoding procedure described above. In one embodiment of the present invention, a platform-independent virtual machine, such as a JAVA VIRTUAL MACHINE, is modified to generate requests for portions of an object using an object ID and an offset. Moreover, in one embodiment of the present

8

invention, processors 102 and 103 are configured to execute special instructions for performing load and store operations involving an object ID and an offset--in addition to normal load and store instructions that use virtual addresses.

[0036] Although the present invention is described with reference to a computer system 100 with two levels of cache, the present invention can generally be used with any single-level or multi-level caching structure. Furthermore, although computer system 100 includes two processors, the present invention can generally be used with any number of processors.

Object Table Entry

[0037] FIG. 2 illustrates an object table entry 202 within object table 109 in accordance with an embodiment of the present invention. As is illustrated in FIG. 2, object table entry (OTE) 202 includes a number of items, such as a physical address 204, which specifies a location in main memory 110 for the corresponding object.

[0038] OTE additionally includes a read-only bit 206. If read-only bit 206 is set, all copies of the corresponding object in any object caches associated with translator 108 are read-only copies. On the other hand, if read-only bit 206 is clear, a copy of the object in an object cache associated with translator 108 may be an "owned" copy or a "modified" copy.

[0039] OTE 202 also contains other bookkeeping information 208, such as the size of the object, which can be used while manipulating the object.

Process of Accessing and Object

[0040] FIG. 3 presents a flow chart illustrating the process of accessing an object in accordance with an embodiment of the present invention. The system initially receives a request to access the object, wherein the request includes an

9

object identifier (OID) that is used to reference the object in the object-addressed memory hierarchy (step 301). Next, the system uses the object identifier in an attempt to access the corresponding object in the object cache (step 302). This attempt either results in a cache hit or a cache miss (step 304).

5   **[0041]** In the case of a cache hit, the system accesses a copy of the object in the object cache (step 307). At this point, the request is satisfied.

     [0042] Otherwise, in the case of a cache miss, the system forwards the object identifier and the corresponding object offset to translator 108 (see FIG. 1) (step 306). Translator 108 then uses the object identifier to lookup OTE 202 in

10   object table 109 (step 308).

     [0043] Next, the system determines if the request is a read request or a write request (step 310). If the request is a read request, the system uses the physical address 204 (and the object offset) to access the object in main memory 110 (step 311). At this point, the request is satisfied.

15   **[0044]** On the other hand, if the request is a write request, the system determines if read-only bit 206 in OTE 202 is set (step 312). If not, the system proceeds to step 311 to access the object in main memory 110, at which point the request is satisfied.

     [0045] Otherwise, if read-only bit 206 is set, the system performs a

20   corrective action to deal with the fact that a write request is directed to a read-only object (step 314). In one embodiment of the present invention, performing this corrective action involves causing a fault handler in the requesting processor to perform the corrective action.

     [0046] Note that a number of corrective actions are possible. For

25   example, the corrective action can involve obtaining a writable copy of the object, clearing the read-only bit to indicate that the object is no longer read-only, and then updating the writable copy of the object with data from the write request. It

10

can also involve updating a remotely located master copy of the object with data from the write request. The corrective action can additionally involve terminating the requesting process because the write request is not allowed. In another example, if the request is directed to a debugging breakpoint, the corrective action can involve pausing the requesting process at the debugging breakpoint and clearing the read-only bit.

Process of Making An Object Read-Only

[0047] FIG. 4 presents a flow chart illustrating the process of making an object read-only in accordance with an embodiment of the present invention. Referring to FIG. 1 and FIG. 4, in order to make an object read-only, the system first sets the read-only bit 206 of the corresponding object table entry 202 to indicate that the object is read-only (step 401). Next, the system executes a read-with-intent-to-only-read (RWITOR) instruction on each cache line of the object.

[0048] The RWITOR instruction can be thought of as a special cache flush instruction that causes any modified version of an object located in any cache in a local cache-coherent domain to be flushed out to main memory 110. When a processor 102 executes a RWITOR instruction on a cache line associated with an object (step 402), the system first determines if an L1 cache 104 associated with processor 102 contains modified or owned copies of the cache line associated with the object (step 404).

[0049] If so, the system writes any modified line back to L2 cache 106, and changes the state of the modified or owned line in L1 cache 104 to the shared state (step 406). Otherwise, if L1 cache 104 does not contain modified or owned lines associated with the object, the system broadcasts the RWITOR request to sibling L1 caches (such as L1 cache 105), which belong to the same local cache-

11

coherent domain (step 408). This causes the sibling L1 caches to write modified lines back to L2 cache 106.

[0050] Note that in one embodiment of the present invention, the system either allows a single owned or modified copy of an object to exist in a single object cache associated with translator 108, or it allows multiple read-only (shared state) copies of the object the exist in multiple object caches associated with translator 108. This ensures that a read-only copy of the object must communicate with the translator to be upgraded to owned or modified state. This is because the read-only copy is not able to receive an upgrade directly from an owned or modified copy in another object cache associated with the translator 108.

[0051] Next, the system broadcasts the RWITOR request to L2 cache 106 (step 410). This causes L2 cache 106 to similarly flush any modified caches lines out to main memory 110.

[0052] In this way subsequent upgrades of the object from read-only status to writable or modified status in any caches associated with translator 108 must go through the translator 108.

Processing a Request From a Remote Processor

[0053] FIG. 5 presents a flow chart illustrating how the system handles a request to make an object read-only received from outside of a local cache-coherent domain in accordance with an embodiment of the present invention. The process starts when a request (for an object or a cache line associated with an object) is received from a remote machine that is located outside of the local cache-coherent domain (step 502).

[0054] In response to this request, the system makes the corresponding object read-only. The system does this by first setting the read-only bit 206 in the corresponding object table entry (OTE) 202 for the object to indicate that all

12

copies of the object in object caches associated with translator 108 are read-only. Next, the system executes an RWITOR instruction on all cache lines associated with the object in all object caches within the local cache-coherent domain (step 504). This causes any modified cache lines associated with the object to be

5   flushed out to main memory (and possibly forwarded to the processor the initiated the RTO operation).

[0055] Finally, the remote machine can obtain the latest modified version of the object (if necessary), can clear its corresponding read-only bit for the object, and can modify the object as desired (step 506).

10   [0056] The foregoing descriptions of embodiments of the present invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not

15   intended to limit the present invention. The scope of the present invention is defined by the appended claims.

13